The effect of truncation in statistical computation

van Reeken, A.J.

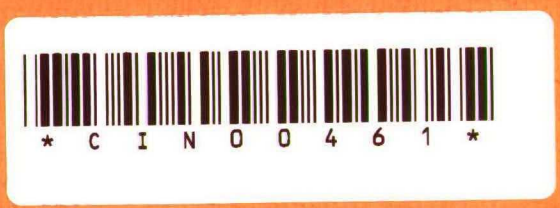
Publication date:
1970

Document Version
Publisher's PDF, also known as Version of record

A. J. van Reeken

The effect of truncation in statistical computation

Research memorandum



TILBURG INSTITUTE OF ECONOMICS
DEPARTMENT OF STATISTICAL COMPUTATION

The effect of truncation in statistical computation

A.J. van Reeken

0. INTRODUCTION

At the time when the computer was introduced for technical and scientific calculations, in general the existing methods for these computations were embodied in the computerprograms without alterations. Since at this phase programming was difficult and time-consuming little attention was given to the efficiency of the method.
After programming-languages, especially FORTRAN, were introduced, more and more attention was paid to this aspect, but the study of new applications, to which study we owe development, was still given preference. Since that time a number of methods for matrix-inversion, roots of equations, eigenvalues, etc, being inefficient, disappeared out of our textbooks. In general we should say this was the case with methods designed for "handcalculation".
The study of errors, mainly due to rounding, in numerical methods has been undertaken for as long as numerical methods existed.
WILKINSON [1] especially has published many results.
However, these results have not quite been embodied in numerical methods used in the fields of statistical and operational research analysis.

It is most significant that in the IBM 1620 European Program Library a number of programs on Linear Programming and on Multiple Regression are available giving results which are sometimes so much altered by truncation errors that the outputted solution is even contradictory. For instance the L.P.-solution does not fulfill the restrictions or the M.R.-solution showes negative residual variances and coefficients of correlation greater than one.

Studying these results one finds that in those cases the routine for matrix-inversion, among other reasons, was a source of error. From an enquiry recently held and not yet finished by the Working Committee on Multiple Regression *), we know that matrix-inversion methods are being used which suffer from more truncation than other methods in use. Also we found one method which was the fastest method on machine X and the most time-consuming one on machine Y.
In both forementioned cases (L.P. and M.R.) scaling, too, seems to be a source of error. In statistical computation, however, the big source of error is truncation accumulated while forming a so-called matrix of sums of squares and cross products.

*) A working committee of the "Working Party on Statistical Computing" of the Dutch Statistical Society.

1. THE CLASSICAL METHOD OF COMPUTING THE VAR-COVAR MATRIX

Let be X a matrix with m columns (variables) and n rows (observations), then the matrix of sums of squares and cross products will be:

$$S = X'X \tag{1}$$

and the vector *) of sums will be:

$$\underline{s} = X'\underline{e} \qquad (\text{where } e_i = 1 \; ; \; i = 1,n \; **) \tag{2}$$

Sometimes $\underline{e}$ is put left of X giving a matrix $G = (\underline{e}/X)$. Multiplication of G'G results in a matrix S of which the first row (and column) is the vector of sums. This is merely a technical convenience that does not have any influence on accuracy.

The matrix of reduced sums of squares and products is to be computed from:

$$\tilde{S} = S - \frac{1}{n} \; \underline{s} \; \underline{s}' \tag{3}$$

If the formula's given above were to be programmed in this way a number of technical difficulties arise, difficulties which one has tried to solve by using floating-point arithmetic.

---

*) Underlined symbols denote columnvectors.

**) By i = 1,n we mean all values of i from 1 through n.

2. FLOATING-POINT ARITHMETIC

Since most of these calculations are being programmed in FORTRAN, integer arithmetic could not be used because this type of arithmetic in FORTRAN seemed to be treated as an aberration (c.f. RODDEN, CACM 10, 3, pg. 180). Calculations in floating-point, however, are giving different results depending on the machine one is using.
KAHAN 3 describes a trick to be used on electronic computers which normalizes floating-point sums before rounding or truncating them. An example of such a machine is the IBM 360 (short word arithmetic). The forementioned trick, however, does not work so well on machines such as the IBM 650 or the IBM 1620, which round or truncate floating-point sums to single precision before normalizing them. (The trick will be described later on).
Floating-point arithmetic is very easy to program even in symbolic coding; with scale there is no trouble at all.
In the meantime a number of people begin to realize that with floating-point arithmetic not all problems can be solved.

Let us look at a simple example. We want to compute:

$$x = -x/\ln(1-y)$$

for different values of y (which values in this example are results of a pseudo-random process). Calculations are being carried out in length of 8 digits. Suppose we find a value of y equal to. 8E-7 *). On the IBM 1620 the answer of 1-y is 1 because the digit 8 is not subtracted from the 9th significant digit of the value 1 and the final result is a division of -x by zero which independently of the value of x gives "minus infinity".

---

*) By .8E-7 we denote $.8 * 10^{-7}$

In fact we know from algebra that z should be approximately equal to x/y which in most cases will be different from "minus infinity". For values 1.OE-7 <y ≤1.OE-8 , TOMPA [4] gave a solution to this problem by programming:

z = -x / ln(0.5 - y + 0.5) in that order.

Computing $\tilde{S}$ (c.f. formula (3)) one gets more or less the same. By building up S and to a lesser extent by building up $\underline{s}$ accumulation is mainly in the positive direction. This is an observation from practical work. The loss of significant digits can best be made clear with an example.

> Suppose we have 10 observations of x: $x_k = 0.01 * k + 99.0$; k = 1,10. One can easily check that (with mantissalength 8) the following totals are obtained: $\Sigma x = 990.55$ and $\Sigma x^2 = 98118.934$. Since the last digit of every squared value is not accumulated, the sum of squares is 0.0045 too low.The subtraction in $\Sigma x^2 - (\Sigma x)^2/n$ results in a corrected sum cf squares equal to 0.004. A correct answer would be 0.00825. For a variable of the form $x_k = 0.01 * k+999.0$ the corrected sum of squares will even be -0.3 and this is negative since the sum of squares now is much too low while the sum is not in error.

This is only an illustration of what may happen. In econometric work e.g. variables like x_k occur in those cases where first logarithms are computed. "The danger in the latter case is generally recognized, referred to as subtraction error, and is most commonly expected to occur in analysis of variance problems where the correction factor is nearly equal to a treatment sum of squares" (c.f. NEELY, CACM 9,7, pg. 496).

3. ERROR-FREE METHODS

In the last five years a number of "error-free" methods and methods for reducing truncation errors have appeared mainly in the ACM Communications.
To begin with we record that in 1960 I learned that of the formulas for computing sums of squares:

$$SS = \Sigma (x_i - \bar{x})^2 \tag{4}$$

$$= \Sigma x_i^2 - n \bar{x}^2 \tag{5}$$

$$= \Sigma x_i^2 - (\Sigma x_i)^2/n \tag{6}$$

formula's (4) and (5) were of theoretical interest and formula (6) was used for computational purpose.
One year later, having experience with a computer we proposed to use (4) and from then on $\Sigma (x_i - \bar{x})^2$ was used for calculation, requiring two passes of the data because storage of the data matrix on the IBM 650 was inefficient.
Later on with the larger IBM 1620 this was no longer necessary and storing was preferred. With this method one gets good results. It is, however, possible to construct cases where this method too gives a wrong answer.

> Suppose $x_k = 0{,}01\ k + 999999{,}0$. Summing x_1 and x_2 results in 1999998,0 and from then on 0,01 * k is truncated leaving a sum of 999999 * n. It will be clear that the corrected sum of squares will be computed as $\Sigma (0{,}01\ k)^2$.

We did not know the above particular example at that time, but being aware of the possibility, it is not very difficult to avoid the wrong answer. If in the second pass the squared deviations are summed one also has to sum the deviations themselves. Then one proceeds as follows.

Let m be the final value of the mean and $\bar{x}$ the computed mean from the first pass, we then get after the completion of the second pass

$$m = \bar{x} + \Sigma(x_i - \bar{x}) /n \tag{7}$$

$$SS = \Sigma(x_i - \bar{x})^2 - \{\Sigma(x_i - \bar{x})\}^2/n \tag{8}$$

which means that a correction is applied to formula (4) in the sense of formula (6).
Although this method results in very accurate statistics, two passes of the data are necessary.

In 1962 WELFORD published a note which appeared in Technometrics [6] . We found this note two years later, ran some test-data with it and were soon convinced about its usefulness. WELFORD gave the method as follows:

$$m_0 = 0;\ m_i = (\frac{i-1}{i})m_{i-1} + \frac{1}{i}x_i\ ,\ i=1,\ n;\ \bar{x} = m_n \tag{9}$$

$$s_0 = 0;\ s_i = s_{i-1} + (\frac{i-1}{i})(x_i - m_{i-1})^2\ ,\ i=1,n;\ SS=s_n \tag{10}$$

He also developed formula's for the corrected sums of products and higher moments, which are based on the same principle. The point is that with this method the matrix Q is build up iteratively and that it only requires one pass of the data. Of all possible ways of programming the above mentioned formula's however, we found that in particular (9) and (10) were not the best available.
The best formula's we found are given below:

$$m_0 = 0;\ m_i = m_{i-1} + (x_i - m_{i-1})/i,\ i=1,n;\ \bar{x} = m_n \tag{11}$$

$$s_0 = 0;\ s_i = s_{i-1} + (x_i - m_{i-1})^2 - (x_i - m_{i-1})^2/i, i=1,n;\ SS=s_n \quad (1$$

In the meantime WOLFE [7] reported:

"In accumulating a sum such as in a numerical integration with a large number of intervals, the sum itself becomes much larger than the individual addends. This may produce a less accurate sum as the number of intervals is increased".

This paradox is exactly what we found in 1961 with formula (6). The more units in the sample the less accurate we could compute the statistics. His solution is:

"Separate variables can be established as accumulators to hold partial sums within various distinct intervals. Thus, the extensive successive truncations are eliminated".

In January 1965 ROSS [8] supplied a similar technique which was easier to implement and needed no reprogramming with respect to change of scale. The principle of WOLFE and ROSS, cascading accumulators, is indeed a good solution. However, the computation is slow because the program has to find out in what particular accumulator the observation on hand has to be counted.
Since we found that from formula (11) and (12) Σx and Σx^2 were more accurately retrieved than when computed directly, we think that WOLFE and ROSS will obtain equally accurate results in less time by applying formula (11) and afterwards multiplying by the number of intervals.

Another approach to this particular problem was communicated by KAHAN [3] , which has already been mentioned. The rounding or truncation in S = S + X could contribute to a loss of almost $\log_{10} N$ significant decimals in S, if there are N numbers to be summed. He states:

"Of course, the simplest and fastest way to prevent such figure-loss is to accumulate S to double-precision".
And further:
"---------that double-precision will soon be universally acceptable as a substitute for ingenuity in the solution of numerical problems".

Besides the fact that evaluation of Σx^2 to double-precision requires four times as much time as to single-precision, and that not all machines have convenient accessibility of double-precision, this statement is a challenge.
His trick, to be used on computers which normalize sums before truncating them, is:

```
1  S = 0.0
2  S2 = 0.0
3  DO 8 I=1, N
4  X = ........
5  S2 = S2 + X
6  T = S + S2
7  S2 = (S-T) + S2
8  S = T
9  CONTINUE
```

Even on an IBM-1620 this works out better than normal summing, although if all digits in S are significant and at a particular addition the exponent is increased, the truncated part is not collected in S2 correctly.

In July 1966 a study of NEELY appeared[9] who compared the robustness of several methods. We come back to that study later on.

In January 1967 GOLDBERG showed [10] that in computing least squares polynomials adjustment of the constant term to correct the accumulated rounding errors is simply to add the sum of the deviations.
$\frac{1}{n} \Sigma(Y-Y^*)$ to it. This remark is similar to formula (7).

In March 1967 we find a method of RODDEN [2] , which is accompanied by a critic on NEELY's study. The method presented by him is an iterative method and resembles the method of WELFORD. The difference is essentially that RODDEN uses integer accumulators.

Since the data are first scaled to integers the summations are also integers, i.e. summations of differences with regard to an integer approximation of the mean. When these approximations have to be increased or decreased because the accumulators overflow, this change is transferred as follows:

s = sum of deviations around approximate mean c

t = sum of squared deviations around approximate mean c

d = entier (s/n)

c'= entier ((s + nc)/n)

t'= t + d (nd - 2s) (Note that d = c' - c

s'= s - nd

The last two statements are according to the definitions:

$$s = \Sigma(x-c') = \sum(x-c) - n(c' - c) \qquad (13)$$

$$t = \Sigma(x-c')^2 = \sum(x-c)^2 - 2(c'-c) \sum(x-c) + n(c'-c)^2 \qquad (14)$$

where c and c' are arbitrary.
Finally the mean and corrected sum of squares can be found with formula's (7) and (8).

Already knowing that in principle WELFORD's suggestion was a correct one, one might arque why RODDEN presents this technique. He therefore raises a number of arguments in his sections 2 and 3 of which the good ones are valid for WELFORD's approach too and the bad ones are contradicting his own statements. We will not discuss these arguments here but will quote RODDEN where he replies to NEELY:

> "As error-free, single pass, faster methods are available this "algorithm" can be "best" only in a highly restrictive sence".

Going back to NEELY we quote the sentences to which RODDEN replied:

> "The best algorithm for computation of the mean is
>
> $$\bar{x} = \frac{1}{n} \Sigma x + \frac{1}{n} \Sigma (x - \frac{1}{n} \Sigma x)$$
>
> which requires two passes on the data. Equivalent results can be obtained by use of the usual formula if coded in double precision.
> Two algorithms gave equally good results in the computation of sums of squares or sums of cross products. If the means are computed accurately (as above), then subtraction of the mean first gives results equal to those obtained by correction of results computed on the basis of an approximate mean."

The formula above is the same as formula (7). Translated to our notation NEELY's statement is: Use formula's (7) and (4), or when having an approximate mean use formula's (7) and (8).

In 1961, as said, we arrived at a similar conclusion after less thorough testing. But the formula of WELFORD was also tested by NEELY who nevertheless arrived at the same conclusion.
This was an invitation to us to scrutinize NEELY's work. Because he published data, formula's and results we could find out that he had programmed WELFORD's formula's (9) and (10) without modification (one of his means showed a negative sign, this is possible with formula's (9) and (10), not with (11) and (12) for the data on hand).
Because NEELY had used a 27 bit wordlength which is approximate 8½ decimal digit we were not able to repeat his work on our IBM-1620 exactly. We chose the following approximation. The testing variables $x_{i,6}$ until $x_{i,10}$ were computed with NEELY's method and decreased with 10.

We then have:

$$x_{i,j} = 0.01 * i + (10^{j-4} - 10);\ j=6,\ 10;\ i=1,n$$

The computations carried out with mantissalength 8 showed similar or better results than obtained with other methods except for variable $x_{i,10}$ which led to wrong answers. This is obvious because it is impossible to compute a 9 significant figure accurately with a mantissalength of only 8. In the 27 bit word used by NEELY this figure is represented correctly.

Finally the practise. LONGLEY [5] gave "An appraisal of least squares programs for the electronic computer from the point of view of the user". For a summary of his work we quote SCHEINOK [11] :
"He shows that in many cases the computer-calculated regression coefficients agreed so little with the actual correct answers - as elicited form a desk calculator where no roundoff was allowed - that the computer results were utterly worthless. In some cases there was no agreement in any digit of the coefficients, and even the signs were reversed. Is is the author's thesis that the different matrix-inversion algorithms in particular were at fault, but other numerical procedures were also to blame".
An indication of this last point is given by LONGLEY [5] when he states:
"All programs tested with the means out first improved in accuracy from three to four digits over the original routine where the means were left in. It must be said that removal of the means before taking the cross-products gives a short reprieve but does not really solve the problem of rounding".
We ran the testproblem given by LONGLEY with two versions of a regression program. One version with the classical method (formula(3)), the other with the modified WELFORD-method (formula(11) and (12)).

The first version produced results with no correct digits, the second version produced results with four to five correct digits. The computation was carried out with a mantissalength of eight digits.

4. CONCLUSION

We think that our formula's (11) and (12) and the other formula's put forward by WELFORD, if programmed correctly, need no double-precision as a substitute. If one wishes to exclude every possible source of error we may recommand KAHAN's approach together with WELFORD's. This gives excellent results, if programmed carefully.
In Appendix I several routines are given to be used in statistical programs.

5. ACKNOWLEDGEMENTS.

I wish to express my appreciation to my collegues of the Computing Centre of the Tilburg School of Economics, Social Sciences and Law for their interest and valuable suggestions and especially to Miss Annette Hoefnagels for her assistence in programming. Further to Brian S.Wilson of Esso Petroleum Co.Ltd. (London) and to J.Kriens of the Tilburg School for their many valuable suggestions in preparation of the manuscript.

6. REFERENCES

[1] WILKINSON, J.H.,: "Rounding Errors in Algebraic Processes". Her Majesty's Stationery Office, London, 1963.

[2] RODDEN, B.E.,: "Error-free Methods for Statistical Computations". Comm. of the ACM, Vol.10, nr.3, (March 1967), pg. 179-180.

[3] KAHAN, W.,:"Further Remark on Reducing Truncation Errors", Comm. of the ACM, Vol. 8, nr.1, (Jan. 1965), pg 40.

[4] TOMPA, H.,: private communication, dated Oct. 19, 1966.

[5] LONGLEY, James W.,:"An Appraisal of Least Squares Programs for the Electronic Computer from the Point of View of the User", JASA 62 , 319 (Sept. 1967) pg. 819-841.

[6] WELFORD, B.P.,:"Note on a Method for Calculating Corrected Sums of Squares and Products", Technometrics IV (1962), pg 419-420.

[7] WOLFE, J.M.,:"Reducing Truncation Errors by Programming", Comm. of the ACM, Vol 8, nr 1 (Jan. 1964), pg 355-356.

[8] ROSS, D.R.:"Reducing Truncation Errors Using Cascading Accumulators", Comm. of the ACM, Vol 8, nr 1 (Jan. 1965) pg 32-33.

[9] NEELY, Peter M.,:"Comparison of Several Algorithms for Computation of Means, Standard Deviations and Correlation Coefficients", Comm. of the ACM, Vol 9, nr. 7, (July 1966), pg 496-499.

[10] GOLDBERG, Morton,:"On the Computation of least Squares polynomials", Comm. of the ACM, iol 10, nr 1, (Jan. 1967), pg 56-57.

[11] SCHEINOK, P.,: Computing Reviews, Vol 9, nr 1, (Jan. 1968), pg 56.

APPENDIX I

A. WELFORDS's approach

```
      SUBROUTINE STAT (X,Y,N,GX,GY,SX,SY,RXY)
      DIMENSION X(1), Y(1)
C     THIS SUBROUTINE EXPECTS ARRAY X AND ARRAY Y OF
C     LENTH N BEING READ IN BY THE MAIN PROGRAM AND
C     THAT THE LAST FIVE PARAMETERS ARE SET TO ZERO.
C     AFTERWARDS THESE FIVE CONTAIN THE MEANS, STAND.
C     DEV. 'S AND THE COEFFICIENT OF CORRELATION.
    1 ZN = N-1
    2 ZN = SORT  (ZN)
    3 DO 14 I = 1,N
    4 ZI = I
    5 XM = Y - GX
    6 YM = Y _ GY
    7 GX = GX + XM/ZI
    8 GY = GY + YM/ZI
    9 PXY = XM * YM
   10 XM * * 2
   11 YM = YM ** 2
   12 SX = SX + XM - XM/ZI
   13 SY = SY + YM - YM/ZI
   14 RXY = RXY + PXY - PXY/ZI
   15 SX = SQRT  (SX)
   16 SY = SQRT  (SY)
   17 RXY= RXY / (SX * SY)
   18 SX = SX/ZN
   19 SY = SY/ZN
      RETURN
      END
```

B. WELFORD's approach together with KAHAN's trick.

We will list the new code for statement 7 and 12, the other statements have to be changed similarly.

```
 7  GX = GX + XM/ZI          becomes  77  GX2 = GX2 + XM/ZI
                                      72  T = GX + GX2
                                      73  GX2 = (GX-T) + GX2
                                      74  GX = T

12  SX = SX + XM - XM/ZI     becomes 121  SX2 = SX2 + XM - XM/ZI
                                     122  T = SX + SX2
                                     123  SX2 = (SX-T) + SX2
                                     124  SX = T
```

C. Routine for building a triangular matrix in vector form of net sums of squares and products A(K) and a vector of means Y(K).

```
C       Y(I) AND A(K) HAVE BEEN CLEARED
C       NOBS = NUMBER OF OBSERVATIONS
C       NVAR = NUMBER OF VARIABLES
        DO 15 L = 1, NOBS
        XN = L
        READ (....) (X(III), III = 1, NVAR)
        DO 5 I = 1, NVAR
        X(I) = X(I) - Y(I)
      5 Y(I) = Y(I) + X(I)/XN
        K = 0
        DO 10 I = 1, NVAR
        P = X(I)
        DO 10 J = I, NVAR
        Q = P * X(J)
        K = K + 1
C***** A(K) = A(I,J), STORED BY ROWS IN UPPER TRIANGULAR FORM
C***** OR ALTERNATIVELY COLUMNS IN LOWER TRIANGULAR FORM.
     10 A(K) = A(K) + Q - Q/XN
     15 CONTINUE
```

D.

```
C
C     ***********************************************************************
C
C     SUBROUTINE  XMAALX
C
C     PURPOSE
C        TO COMPUTE BY ITERATION AND RELATIVELY ACCURATE
C          - A VECTORWISE UPPER TRIANGULAR MATRIX OF NET SUMS OF SQUARES
C            AND PRODUCTS
C          - A VECTOR OF MEANS
C
C     USAGE
C         CALL  XMAALX(WN,NVAR,Y,A,X,N1,N2)
C
C     DESCRIPTION OF PARAMETERS
C         WN   -OBSERVATION NUMBER OF N1-TH ROW OF X
C         NVAR -TOTAL NUMBER OF VARIABLES(EQUALS NUMBER OF ELEMENTS
C               IN A ROW OF X)
C         Y    -THE MEANS COMPUTED UP TO THIS STEP UPON ENTRY  AND
C               THE MEANS COMPUTED UP TO AND INCLUSIVE THIS STEP UPON
C               EXIT.BEFORE FIRST ENTRY SET Y(I)=0.0 FOR I=1,NVAR.
C         A    -THE VECTOR OF NET SUMS OF SQUARES AND PRODUCTS (THE
C               UPPER TRIANGULAR PART OF THE MATRIX) COMPUTED UP TO
C               THIS STEP AND REFINED BY THIS STEP.BEFORE FIRST ENTRY
C               SET A(I)=0.0  FOR I=1,(NVAR*(NVAR+1)/2).
C         X    -THE OBSERVATION ROWS WN,WN+1,...,WN+N2-N1 OF NVAR
C               ELEMENTS EACH.THE CONTENTS OF X ARE DESTROYED BY THIS
C               ROUTINE
C         N1   -INDEX OF FIRST ROW IN X TO BE TAKEN
C         N2   -INDEX OF LAST  ROW OF X TO BE TAKEN
C
C     REMARKS
C         WN AND N1 NEED NOT TO BE EQUAL. X CAN NOW BE INPUT FROM A
C         BACKINGSTORE IN BLOCKS OF SAY 10 ROWS(WITH N1=1,N2=10 AND
C         WN=1,11,21...)OR FROM CARDS ONE ROW AT A TIME(WITH N1=1,N2=1
C         AND WN=1,2,3,...)
C
C     NO SUBPROGRAMMES NEEDED
C
C     METHOD
C         FOR REFERENCE SEE (1) AND (2)
C         (1) WELFORD,B.P. ''NOTE ON A METHOD FOR CALCULATING CORRECTED
C                            SUMS OF SQUARES AND PRODUCTS'',
C                          TECHNOMETRICS IV(1962),419-420.
C         (2) VAN REEKEN,A.J. ''LETTER TO THE EDITOR DEALING WITH
C                              NEELY'S ALGORITHMS FOR THE COMPUTATION
C                              OF VARIOUS STATISTICS'',
C                             COMMUNICATIONS OF THE ACM, VOL 11,NO.3,
C                             (MARCH 1968),149-150.
C
C                                                    A J VAN REEKEN MEI 68
C     ***********************************************************************
C
```

```
      SUBROUTINE  XMAALX(WN,NVAR,Y,A,X,N1,N2)
      DIMENSION  Y(1),A(1),X(1)
      TEL=WN
      DO 3 L=N1,N2
      IBASE=(L-1)*NVAR
      DO 1 K=1,NVAR
      INDEX=IBASE+K
      X(INDEX)=X(INDEX)-Y(K)
    1 Y(K)=Y(K)+X(INDEX)/TEL
      K=0
      DO 2 I=1,NVAR
      INDEX=IBASE+I
      Q=X( INDEX )
      DO 2 J=1,NVAR
      INDEX=IBASE+J
      P=Q*X( INDEX )
      K=K+1
    2 A(K)=A(K)+P-P/TEL
    3 TEL=TEL+1.
      RETURN
      END
```

E. Modification

Carol Herraman [1] showed that the routine listed under C and subroutine XMAALX could be programmed more compactly by storing columnwise in upper triangular form or alternatively by rows in lower triangular form.
We give the essence of her solution below and the reader may verify that it needs only three instead of four Do-loops:

```
      DØ  10  L = 1, NØBS
      XN = L
      K = 0
      DØ  10  L = 1,NVAR
      X(I) = X(I) - Y(I)
      P = X(I)
      Y(I) = Y(I) + P/XN
      Q = P * X(J)
      K = K + 1
C**** A(K) = A(I,J)
      A(K) = A(I) + Q - Q/XN
   10 CØNTINUE
```

Reference:

[1] CAROL HERRAMEN : "Algorithm AS12, Sums of Squares and Products Matrix", Applied Statistics, Vol. 17, (1968), nr. 3, pg 289-292.

E.I.T. 1970